INTEGRATION, the VLSI journal 112 (2027) 102912

Contents lists available at ScienceDirect

Integration

journal homepage: www.elsevier.com/locate/vlsi

FPGA-HLS implementation and comparative analysis of fixed- and floating-point matrix inversion methods for least squares computation in compressive sensing applications

Susithra N. [a,*], Rajalakshmi K. [b]

[a] Department of Electronics and Communication Engineering, PSG Institute of Technology and Applied Research, Coimbatore, Tamilnadu, 641062, India
[b] Department of Electronics and Communication Engineering, PSG College of Technology, Coimbatore, Tamilnadu, 641004, India

ARTICLE INFO

Keywords:
QR decomposition
Modified gram-schmidt
Matrix inversion
High-level synthesis
Gaussian elimination
Least squares

ABSTRACT

Matrix inversion is a computationally complex operation used in numerous scientific and engineering domains, from signal reconstruction to control systems. One such application is the Least Squares (LS) problem in Orthogonal Matching Pursuit (OMP), a Compressive Sensing (CS) reconstruction algorithm. Matrix inversion can be performed through QR decomposition (QRD) or the Gaussian Elimination (GE) method. This study investigates the performance of different factorisation algorithms and their effectiveness through High-Level Synthesis (HLS). Through directive-based loop and control optimisations in HLS, the study evaluates latency, resource usage, and throughput. The investigations for both fixed and floating type representation, for different matrix sizes, are reported on a Zynq xc7z020-clg484-1 FPGA device. The optimized floating-point implementation achieves 8.8% higher operating frequency, 67% lower latency, approximately three times higher throughput, and 75% lower numerical error than the fixed-point implementation, while the fixed-point implementation significantly reduces hardware resource utilization, making both implementations suitable for compressive sensing applications with different performance and resource requirements. The optimized Modified Gram-Schmidt (MGS) method and Incremental GE method have better execution time and resource utilization as compared to other classical methods in previous studies, amidst trade-offs. The optimized inversion techniques apply to wireless communications and signal processing applications where resource constraints and precision trade-offs are critical.

1. Introduction

In the field of digital signal processing, wireless communication and machine learning applications, various operations such as adaptive filtering, channel inversion in MIMO systems, beamforming, precoding, image processing, and linear regression involve matrix factorisation and matrix inversion. These operations require efficient and real-time computations. Performing matrix inversion using hardware is complex because of the data interdependencies, involvement of division, continuous memory access for large matrices, challenges in adaptability to changing matrix sizes, precision limitations for the arithmetic operations, time consumption for design, and execution time complexity and cost. Also, the efficiency, accuracy, and numerical stability of various matrix inversion techniques depend on the respective mathematical foundations or the algorithms behind them. Hence, researchers have continually explored and developed innovative ways to leverage matrix inversion for improved signal recovery and processing in resource-constrained real-time scenarios.

Compressive Sensing (CS) and recovery have gained research attention in areas including Digital Signal Processing (DSP) and wireless communication. CS and recovery have wide applications, including speech and music compression, audio processing, and acoustic signal analysis. It is a technique used to efficiently acquire and represent signals, such as images or audio, by using fewer samples than traditional methods. This is particularly useful in scenarios where signal acquisition is resource-intensive, such as in medical imaging, remote sensing, and wireless communication. CS exploits the fact that many signals have a sparse representation in some basis or domain. The main challenge of CS lies in recovering the original signal accurately from the compressed measurements. In CS, this is achieved through reconstruction algorithms

* Corresponding author.
E-mail addresses: nsusithra.6@gmail.com (S. N.), krl.ece@psgtech.ac.in (R. K.).

https://doi.org/10.1016/j.vlsi.2026.102912
Received 15 March 2026; Received in revised form 18 September 2026; Accepted 10 October 2026
Available online 12 October 2026
0167-9260/© 2026 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

like Orthogonal Matching Pursuit (OMP) [1] that utilize mathematical optimisation techniques. Matrix manipulation is essential in CS reconstruction. Reference [2] reviews and simulates the reconstruction of an audio signal using OMP. While implementing OMP, solving the Least Squares (LS) problem is complex, and it is the major hardware resource-consuming step. The existing architectures for the implementation of OMP [3] mostly differ by their approach towards the solution of the LS problem. Also, in applications like medical or audio signal processing, as well as in the recovery process, the nature of the signal is such that the signal requires a high sampling rate. They involve real-time processing and noise removal. The computation time demands parallel processing requirements. Hence, the recovery algorithm involving the LS problem poses a high computational demand.

An efficient algorithm for performing matrix inversion is required. One reason for the high complexity of the LS problem is the involvement of matrix inversion. Matrix inversion becomes crucial in these algorithms as it involves solving linear systems of equations to find a sparse signal that satisfies the measured constraints. The matrix inversion and, in turn, the LS solution determine the numerical stability and accuracy of the reconstruction, especially when the process is iterative. In CS, the measurement matrix helps in converting the high-dimensional signal to a lower-dimensional compressed space and its size and properties are based on the specific application. Different types of measurement matrices, like random matrices, can be used for audio signals [4]. In scenarios where the measurement matrix is not square or full-rank, the matrix pseudo-inversion method called the Moore-Penrose pseudo-inverse is used. These techniques leverage matrix inversion principles to find approximate solutions to underdetermined and over determined systems of equations. Hence, the efficiency and accuracy of CS and reconstruction algorithms depend on how effectively matrix inversion is carried out.

Matrix inversion can be performed by many matrix factorisation techniques such as Gauss-Jordan method, Gaussian Elimination (GE), Cholesky Decomposition, Lower-Upper (LU) Decomposition, QR Decomposition (QRD), Singular Value Decomposition and Eigen Decomposition. These techniques involve complex hardware architectures. Yet, many of the algorithms, when mapped to architecture, have high hardware complexity and require considerable resources. Efficient hardware implementation is essential by exploiting parallelism and other optimisations to speed up the computation. The hardware architectures are implemented in the FPGA or an ASIC design to investigate the area, time, and power utilized.

To overcome the above-mentioned challenges, many studies have been done. The previous studies have shown modifications in the algorithm [5–7] through numerical methods and approximations. Some researchers have performed architectural transformations [8] to overcome the resource utilization challenges at lower abstraction levels. There are studies demonstrating approaches in implementing the matrix inversion module in different FPGA devices or as an ASIC design using IP cores [9]. Recently, to implement such complex modules, tools like Vivado/Vitis HLS and Intel HLS provide documentation that explores the prospects of loop optimisation options through pragmas and directives.

This paper investigates the performance of matrix inversion techniques through different algorithms based on QRD and GE techniques. In order to overcome the hardware complexity and the time overhead required to accomplish architectural transformations at lower abstraction levels, this paper deploys the loop optimisation techniques in HLS. The authors demonstrate the loop optimisations and control flow level optimisations for factorisation and inversion. Such optimisations at the high-level coding can help in the resource reduction and improvement of throughput. It is practically easier and requires less design time, thereby enhancing the speed of mapping an algorithm to an efficient architecture. Resources consumed, latency and throughput for implementations with and without optimisation are compared with the data from previous works. The paper also analyses the performance of the above-mentioned algorithms for both fixed- and floating-point representation. Besides, the scalability of matrices for the implementations is reported. All the results are pertaining to the Zynq XC7Z020-CLG484-1 FPGA device.

In this paper, the section 2 elaborates on the learnings from the literature, section 3 gives a glimpse of the QRD and GE algorithms, section 4 is the methodology is that explains the strategies adopted to perform optimisations using HLS tool and section 5, Results and Discussion, explains the performance analysis and comparison with previous studies.

2. Literature Survey

Several previous studies have explored the use of different algorithms for matrix inversion and addressed their suitability for various applications. In Ref. [9] Matrix inversion GUSTO core generator is used to perform matrix inversion, and a trimming feature is used for optimisation to reduce the resources. The architecture uses DSP48 FPGA resources instead of LUTs. In Ref. [10] Gauss-Jordan elimination method is studied using MATLAB VHDL code generator, and the results suggest that normalization reduces the resources used in the design. The analysis is done for single-precision floating-point type data, and the performance of double precision is not available.

Some research works focus on systolic cell architectures for the matrix factorisation, and in one such work, a line-by-line systolic array architecture with fixed-point serial optimisation based QRD is designed as a solution to a preprocessing algorithm for a MIMO detector and reports a high throughput [11]. In another work systolic array and CORDIC algorithm-based Givens Rotation (GR) algorithm is synthesized for fixed-point data type, and folding transformations are adopted as a means of architectural optimisation [12]. Synthesized resource estimations are observed to be high, and no further resource utilization after implementation and optimisation is reported. Reference [13] is also a systolic array based; additionally, a piecewise polynomial approximation for computing the reciprocal square root is suggested for time-critical MIMO applications. A matrix scalability analysis is also reported. In the work proposed in Ref. [14], a high-throughput QRD with CORDIC Core Generator using the GR method is presented, which utilizes fixed-point 2-D systolic array pipelined processing elements. It is suggested that there is scope for improvement if a pipelined version of DSP48 is used, but at the cost of a moderate increase in the latency and also cites that this is suitable for small matrices.

HLS of iterative-pipelined and parallel pipelined Gram-Schmidt based QR factorisation are presented in Ref. [8] for complex matrices. In Ref. [15], both floating point as well as fixed point Gram-Schmidt QRD are synthesized for a 100x100 matrix, but report an error up to 3%. In another article, the GR method using a proposed HUB-floating point QRD is implemented in Xilinx ISE, with a CORDIC rotator [16]. It is also suggested that the units can be used for designing highly parallel, low-cost iterative QRD units. Reference [17] also proposes a CORDIC-based QRD for MIMO applications. Similar to Ref. [8], the authors of [18] propose the applicability of Systolic hardware and hybrid piecewise polynomial approximation-based QRD with reduced resources in a coprocessor used in a hardware and software codesign paradigm. LDL factorisation and a recursive algorithm for inversion are used in Ref. [5] applicable in control engineering. In the article, Hermitian and non-Hermitian matrix inversion are solved by deploying the parallel pipelining mechanism.

The diverse methods for matrix factorisation determine the accuracy and hardware complexity in applications such as the LS problem in the OMP algorithm. In Ref. [6], the authors have proposed an Incremental GE (IGE) Method. The same authors in Ref. [7] have performed a partial evaluation of QRD for accomplishing lower complexity in the OMP implementation. Modified QRD is suggested as it is less time consuming than other QR methods. Alternatively, in Ref. [19], the authors approach two more methods for inversion, namely Singular value decomposition

and Recurrent Neural Network method, occupying a decent number of BRAMs and DSP units, considering the hardware complexity. It is reported in Ref. [20] that high throughput and area reduction are observed when distributed arithmetic for multiplication and systolic arrays for the LS problem are used in the OMP.

In sparse signal processing applications that employ CS and reconstruction, solving the LS problem is a key step. LS problem involves the matrix factorisation and inversion, and this can be approached by QRD or GE methods to minimize the computation cost. In IoT edge applications, reduction in area and the cost of computation also helps reduce the power. An algorithm to approach QRD is the use of the Gram-Schmidt process. In order to accelerate the computations involved in the QRD or the GE method of matrix inversion, and also to achieve high throughput, an efficient hardware design is required. Implementing suitable hardware can exploit parallelism and hardware transformation techniques. Though this enables efficient matrix operations, it requires more design time and effort. In Ref. [21] the authors have developed a low-area, high-throughput OMP for ECG reconstruction, with incremental Gram–Schmidt (IGS)-based QRD and detailed resource Vs throughput analysis on FPGA. Recent works have also used HLS based design strategy, for instance Ref. [22], discusses matrix inversion on FPGA using a recursive Gram matrix inversion, focusing on HLS pragma strategies for latency and throughput.

Matrix inversion in LS problem is particularly applicable to compressive sensing-based channel estimation in MIMO systems, where OMP is widely employed to recover sparse channel information from a limited number of pilot measurements. Recent studies have also demonstrated that OMP-based channel estimation provides improved estimation accuracy over conventional least-squares methods in IRS-assisted MIMO systems, highlighting the effectiveness of compressive sensing for sparse channel recovery [23].

Hence, the main objectives of this paper are.

- Investigation of HLS of optimized matrix inversion methods that perform well when scaling up the matrix for different data types (fixed, single and double precision floating point) and word widths. The area, timing and throughput trade-offs of the matrix inversion process are analysed.
- Application of directive-driven loop and control flow optimisations in the HLS tool in generating efficient hardware architectures with reduced latency and resource utilization.
- Comparison of the results obtained with the previous works on matrix inversion architectures and implementations.
- Demonstration of the suitability of QRD and GE methods for processing large matrices in the LS problem of the OMP for CS reconstruction.

This article demonstrates the outcome of a combination of high-level language programming and model-based design for the FPGA implementation of fixed-point and floating-point matrix inversion algorithms. The article pays special attention to the trade-offs among the area, cost, matrix range, performance and precision aspects.

3. Background

3.1. Compressive sensing and reconstruction

The Compressive Sensing for the acquisition of compressive sampled signals is

$$\boldsymbol{y} = \widehat{\boldsymbol{\phi}}\boldsymbol{s} \tag{1}$$

The OMP algorithm solves the least-squares equation to estimate $\widehat{s}$. The least-squares equation is solved with the help of the Moore–Penrose equation. The recovery using the OMP algorithm takes the measurement matrix $\widehat{\boldsymbol{\phi}}$ and the measurement vector $\boldsymbol{y}$ as input and generates an estimation $\widehat{\boldsymbol{s}}$. $\boldsymbol{s}$ is the sparse representation of $\boldsymbol{x}$ which is the original signal, and $\widehat{\boldsymbol{s}}$ is an estimate of sparse $\boldsymbol{s}$. In OMP algorithm, $\widehat{\boldsymbol{\phi}}^{+}$ which is the pseudoinverse of matrix $\widehat{\boldsymbol{\phi}}$ is computed and multiplied by $\boldsymbol{y}$ to estimate $\widehat{\boldsymbol{s}}$.

$$\widehat{\boldsymbol{s}} = \widehat{\boldsymbol{\phi}}^{+}\,\boldsymbol{y} \tag{2}$$

where, $\widehat{\boldsymbol{\phi}}^{+} = \left(\widehat{\boldsymbol{\phi}}^{T}\,\widehat{\boldsymbol{\phi}}\right)^{-1}\left(\widehat{\boldsymbol{\phi}}^{T}\right)$

The computation of the inverse of this underdetermined matrix is computationally expensive. This is done by QR factorisation, **R** is an upper triangular matrix, and hence pseudo-inverse obtained is

$$\widehat{\boldsymbol{\phi}}^{+} = \mathbf{R}^{-1}\,\mathbf{Q}^{\mathbf{T}} \tag{3}$$

3.2. QR decomposition method for CS reconstruction

In the QRD method, a matrix to be inversed (**A**) is factorized into two components known as an orthogonal matrix (**Q**) and an upper triangular matrix (**R**). The orthonormal basis of the matrix **A**'s column space is represented by the **Q** matrix, while the matrix **A**'s row space is described by the **R** matrix. The inversion of **R** is usually more stable than directly inverting the original matrix **A**. This is because **R** tends to have more favourable numerical properties. For instance, **R** being upper triangular has dominant diagonal elements that reduce the amplification of small errors during inversion. Also, this ensures minimal change in inversion output when there is a change in original input, thereby making the inversion process stable. To find the inverse of matrix **A** by QRD, factorize it as,

$$\mathbf{A} = \mathbf{QR} \tag{4}$$

$$\mathbf{A}^{-1} = \mathbf{R}^{-1}\mathbf{Q}^{-1} \tag{5}$$

$$\mathbf{A}^{-1} = \mathbf{R}^{-1}\mathbf{Q}^{\mathrm{T}} \tag{6}$$

Because **Q** is an orthogonal matrix, the transpose of **Q** is used as the inverse of **Q** and also contributes to numerical stability. It is not so complex to compute $\mathbf{R}^{-1}$, because **R** is an upper triangular matrix, which means that its inverse is also an upper triangular matrix. Also, the diagonal elements of **R** are positive, which makes it easy to compute the inverse of **R** with fewer errors.

Due to its lower computing complexity and increased numerical stability, QRD has a variety of uses in DSP, solving linear system of equations, processing eigen vectors and eigen values [24], least squares fitting, numerical optimisation, and signal processing applications like echo cancellation in an adaptive filter, MIMO equalization and beamforming systems [25].

There are different methods of QRD, such as the Gram-Schmidt (GS) method, the House Holder Transformation (HHT) method, and the Givens Rotation (GR) method. The GS method has 3 variants, namely, Classical Gram Schmidt (CGS), Non-normalized Gram-Schmidt (NGS) and the Modified Gram-Schmidt (MGS) methods. Traditional methods of QRD, like the NGS and CGS, have a complexity of $O(n^3)$, but more efficient algorithms, such as MGS, GR method and HHT [26] can reduce this complexity. The choice of a QRD method will ultimately depend on the application and its requirements, such as accuracy, numerical stability, and computational efficiency.

Gram-Schmidt Orthogonalization is one of the earliest and simplest methods for QRD. The upper triangular matrix **R** is computed by projecting the original matrix **A** onto the orthogonalized columns after sequentially orthogonalizing the columns of the matrix to produce the **Q** matrix [15]. It is a classical algorithm that sequentially orthogonalizes a set of vectors to obtain an orthonormal basis. The CGS algorithm is the simplest approach for QRD, but can suffer from numerical instability due to the accumulation of errors during the orthogonalization process. Whereas, the NGS and MGS algorithms address some of the stability

issues by normalizing the vectors after each orthogonalization step.

GR is another numerically stable technique for QRD that involves computing the **Q** matrix by the classical method. Whereas, an upper triangular matrix **R** is computed by applying a sequence of Givens rotations to the original matrix **A** [12]. In HHT method for QRD, the original matrix is transformed into an upper triangular matrix **R** using a succession of Householder reflector matrices that help to zero out the matrix components by performing reflection of the matrix about hyper-planes. The **Q** matrix is calculated simultaneously. The HHT and GR are alternative techniques that provide numerically stable QRD by minimizing the error accumulation.

The Gram-Schmidt method is more numerically stable than the original approach by re-orthogonalizing the matrix **A**'s columns after each step [6]. So, it is advantageous in terms of numerical stability, preserving orthogonality and easier implementation. Algorithm 1 describes the MGS method for finding the inverse of a matrix.

Algorithm 1.

Input: Matrix $\boldsymbol{A}$
Output: $\boldsymbol{A}^{-1}$
Procedure:
1. Initialize null matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$
2. for $j = 1$ *to* n, do:
  Set $a_j = j^{th}$ column of $\boldsymbol{A}$
  for $i = 1$ *to* $j-1$, do:
    $\boldsymbol{r}_k = \langle \boldsymbol{a}_j^T * \boldsymbol{q}_k \rangle$ for $k = 1$ *to* n
    $\boldsymbol{r}_j = \boldsymbol{r}_j - (\boldsymbol{r}_k * \boldsymbol{q}_k)$
  Set $\boldsymbol{q}_j = \boldsymbol{r}_j / \|\boldsymbol{r}_j\|$ (normalized j^{th} column of $\boldsymbol{Q}$)
Return $\boldsymbol{Q}$ and $\boldsymbol{R}$
3. To Find $\boldsymbol{R}^{-1}$
  Initialize $\boldsymbol{R}^{-1} = 0$
  for $j = 1$ *to* n, do:
    for $i = 1$ *to* $j-1$, do:
      for $k = 1$ *to* $j-1$, do:
        $\boldsymbol{R}_{ij}^{-1} = \boldsymbol{R}_{ij}^{-1} + \boldsymbol{R}_{ik}^{-1}\boldsymbol{R}_{kj}$
    for $k = 1$ *to* $j-1$, do:
      $\boldsymbol{R}_{kj}^{-1} = -\boldsymbol{R}_{kj}^{-1}/\boldsymbol{R}_{jj}$
  $\boldsymbol{R}_{jj}^{-1} = 1/\boldsymbol{R}_{jj}$
4. Find $\boldsymbol{Q}^T$
5. $\boldsymbol{A}^{-1} = \boldsymbol{R}^{-1}\ \boldsymbol{Q}^T$

The implementation of QRD can be computationally intensive, especially for large matrices. This is because, the process involves applying several matrix operations, such as matrix multiplications and orthogonal transformations. The operations consume more time and hardware. Moreover, QRD can be numerically unstable, particularly when dealing with ill-conditioned matrices such as matrices with extreme eigenvalues, near singular matrices or matrices with nearly linearly dependent columns, leading to numerical inaccuracies. To address these issues, modified efficient algorithms and numerical techniques are employed. In a real-time scenario like audio signal recovery via compressed sensing, the measurements received are frequently highly correlated, redundant or ill-conditioned. This necessitates the use of QRD in CS reconstruction. The redundancy in the measurements can be utilized by QRD to effectively recover the original signal from fewer measurements.

3.3. Gaussian Elimination method and incremental Gaussian Elimination method

Another computationally simpler method to obtain the pseudo-inverse is the GE method. The GE is an alternative method used for matrix factorisation. It is a direct method for solving systems of linear equations by transforming the original coefficient matrix into an upper triangular form through a sequence of row operations or row transformations. These operations involve adding or subtracting multiples of rows to eliminate variables and create zeros below the diagonal. Unlike the QRD method, GE does not factorize the matrix explicitly. The estimate in OMP algorithm using $\widehat{\boldsymbol{\phi}}^{+}$ which is the pseudoinverse of matrix $\widehat{\boldsymbol{\phi}}$ is computed as follows,

$$\widehat{s} = \left(\widehat{\boldsymbol{\phi}}^T\ \widehat{\boldsymbol{\phi}}\right)^{-1}\ \left(\widehat{\boldsymbol{\phi}}^T\right) y \tag{7}$$

$$\widehat{s} = \left(\widehat{\boldsymbol{\phi}}^T\ \widehat{\boldsymbol{\phi}}\right)^{-1}\ \left(\widehat{\boldsymbol{\phi}}^T\ y\right) = \mathbf{G^{-1}Y} \tag{8}$$

where G is a symmetric square matrix and Y is a column vector. The Augmented matrix [**G Y**] can be transformed to [**U Z**] by performing row transformations such that **U** is an upper triangular matrix. Again, the inverse of an upper triangular matrix is computationally simpler than the direct inverse of the measurement matrix.

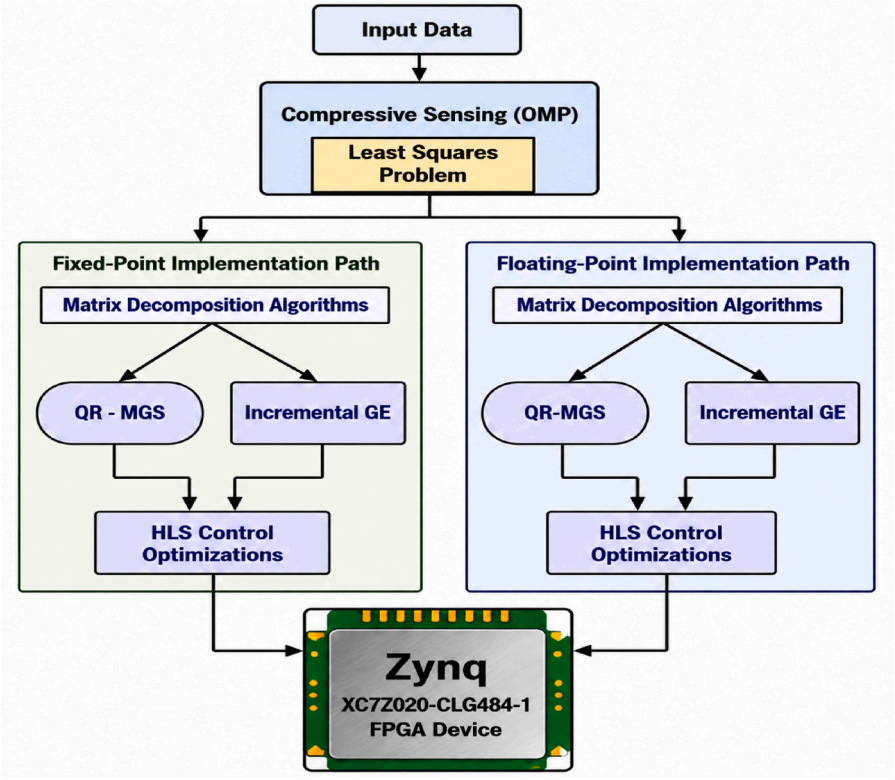
$$\widehat{\boldsymbol{s}} = \mathbf{U^{-1}Z} \tag{9}$$

GE has a computational complexity of $O(n^3)$, where n is the number of columns in the matrix. This makes it less efficient for large matrices [7]. It is suitable for both square and non-square matrices. However, it can become numerically unstable for ill-conditioned matrices due to the accumulation of round-off errors during the elimination process by the row transformations.

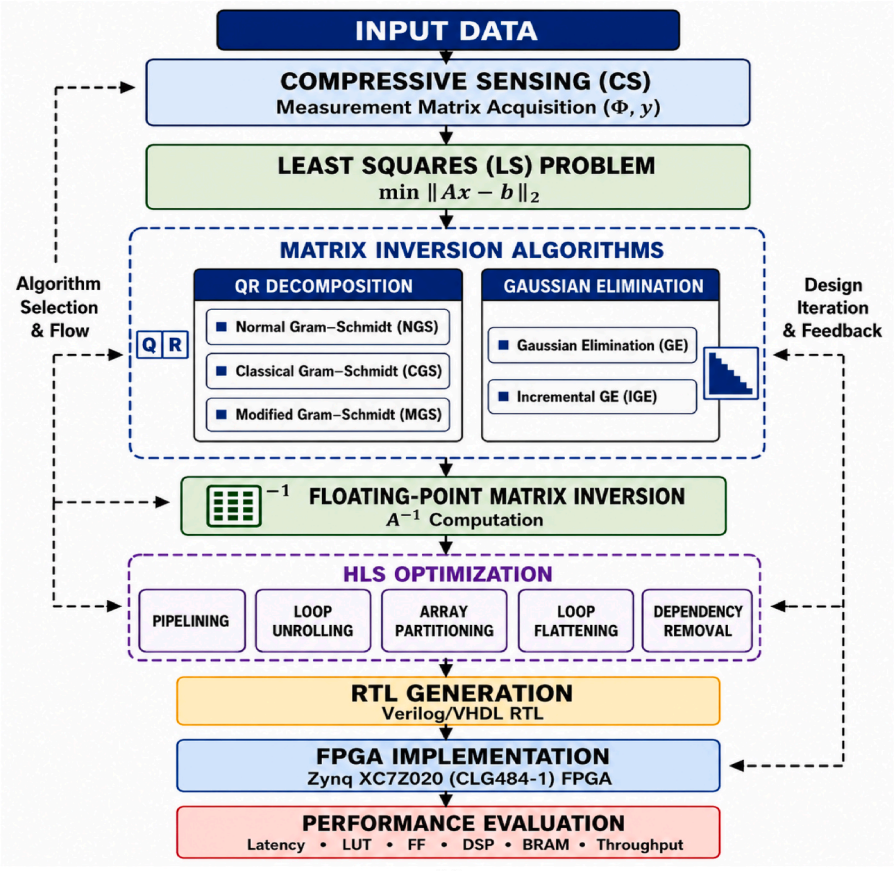
IGE method, or the rank one update method, is a variation of the standard GE algorithm. It is designed to improve efficiency, especially in real-time applications like CS recovery, involving small changes in the matrix dynamically. The IGE method is divided into four major steps. In step 1, the i^{th} column of the matrix **A** is modified except for the element $\mathbf{A}_{ii}$. In step 2, all the elements of the i^{th} row are nullified except the elements $\mathbf{A}_{i,i}$ and $\mathbf{A}_{i,i+1}$. In this step, multiplication constants α values used for row transformation are saved to memory. In step 3, i^{th} row of **A** gets its final form and constant β used to divide the row values to make them to 1 before nullification is computed and saved. The γ matrix consisting of the $\alpha_{i,j}/\beta_i$ is computed in step 4. This uses registers for storing the intermediate result in computations, rather than toggling the registers for every iteration when the value changes, hence reducing the dynamic power caused by toggling or switching activities. The algorithm has $O(n^2)$ complexity, unlike the conventional GE method. Thus, showing superior performance compared to the GE method. This also makes pipelining possible.

4. Methodology

In the literature, realisation of computationally intensive matrix inversion for the OMP based CS reconstruction application has been explored to improve and establish optimal architectures. The challenge of hardware complexity is more in case of real-time applications with dynamic, sparse and scalable matrices. This motivates us to investigate the optimisation options to reduce the design time and effort, at the same time, improve the area, timing and throughput via HLS design tool. QRD and GE algorithms were chosen due to their computational architectures being very well-suited for implementation using HLS. Both of these algorithms consist of regular, deterministic iterations with arithmetic computations which are suitable for mapping to hardware using HLS directives. The algorithms have predictable memory accesses and can be easily pipelined, unrolled, partitioned, and executed in parallel. Moreover, the QRD procedure makes matrix inversion procedures become operations on upper triangular matrices, which reduces the computational complexity. All these factors make it possible to create efficient RTL architecture by the HLS compiler, reducing latency and resource requirements and saving considerable time for hardware design in comparison with conventional RTL implementation. Also, the performance in fixed-point and floating-point data types is analysed. In this paper, the improved performance due to the use of pragmas at appropriate loop and control flow positions has been demonstrated. The overall architecture of proposed system is illustrated in Fig. 1(a) and the HLS based methodology is depicted in Fig. 1(b).



(a)



(b)

Fig. 1. **(a)** proposed hardware architecture for HLS-Based matrix inversion **(b)** proposed methodology for HLS-Based matrix inversion.

4.1. HLS design flow

The algorithm MGS method and the IGE method for matrix inversion are first written in high-level languages, C and C++. Each implementation consists of three principal modules: a testbench and a top function having the algorithmic module that performs the desired inversion function.

- Top Function: Performs matrix inversion by invoking the selected QRD or GE routine and generating the inverse matrix.
- Algorithm Module: Implements the computational block corresponding to MGS, CGS, NGS, GE, or IGE.
- Testbench: Generates input matrices, invokes the top function, compares the synthesized output with software-generated reference results, and reports numerical error.

The source code and testbench are compiled using the GCC compiler. The standard HLS design flow converts C and C++ code into RTL-level HDL code through synthesis, which generates the RTL-level Verilog code for inversion. Further, after verifying the functionality by C/RTL co-simulation, RTL implementation with synthesis, place and route options is performed before exporting into an IP block. The top function is later synthesized, implemented and used to generate Intellectual Property (IP) blocks. This HLS design flow is given in Fig. 2. The directives

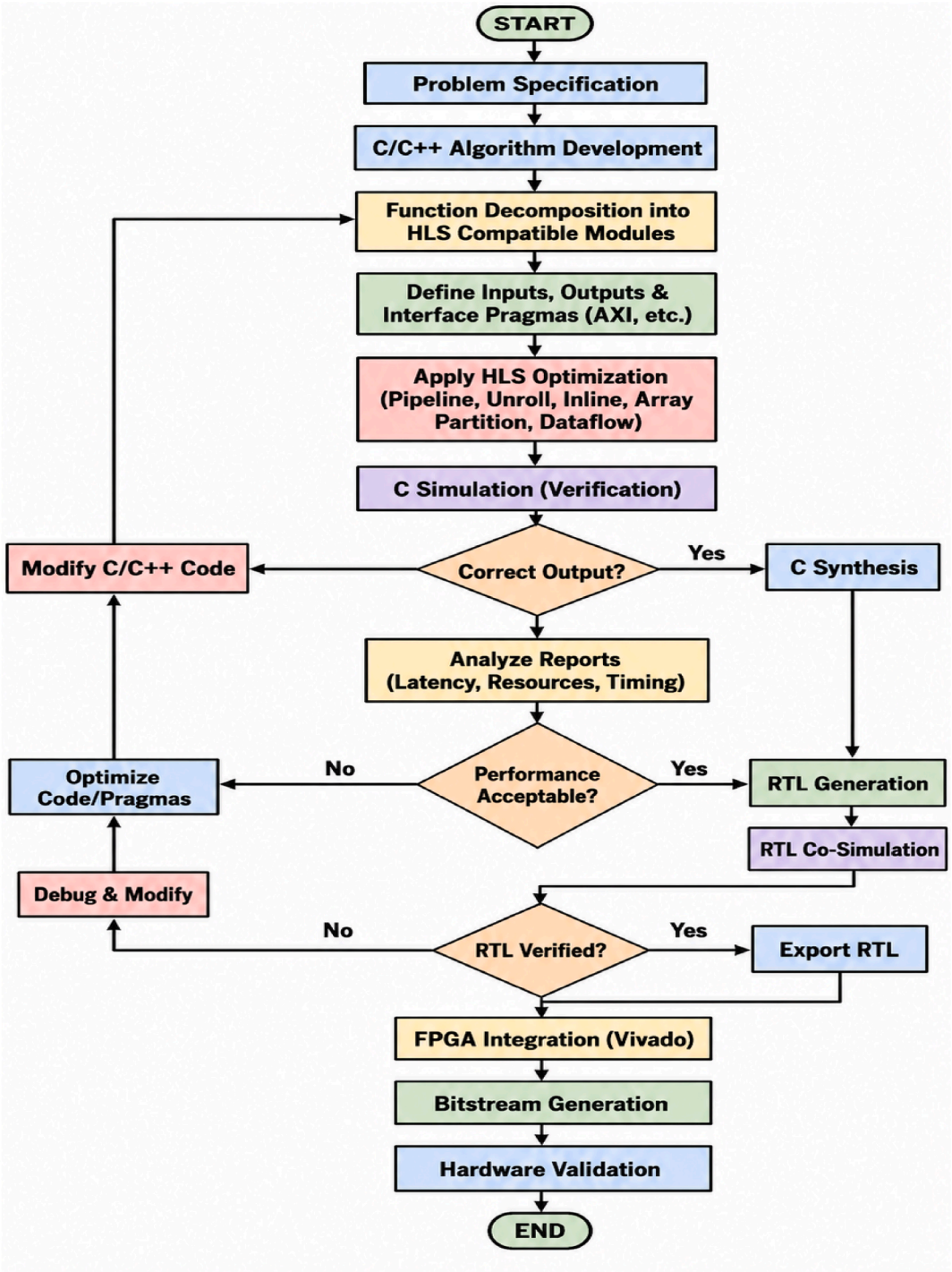


Fig. 2. C/C++ organization of the proposed HLS implementation.

are inserted in the program before the synthesis.

4.2. Hardware optimisation strategies followed in the matrix inversion implementation

Optimisation techniques in Vitis/Vivado HLS offer several advantages that significantly enhance the hardware performance. The techniques, such as loop unrolling, pipelining, and loop flattening, improve code execution efficiency and reduce overall execution time. Pipelining is a technique that allows for the concurrent execution of loop iterations, maximizing hardware utilization and improving throughput. It enables the efficient utilization of functional units within the processor by overlapping the execution of instructions from different iterations. Loop unrolling is a technique where the compiler duplicates loop iterations to reduce loop overhead and improve execution speed. By unrolling loops, the number of loop control instructions decreases, resulting in improved performance due to reduced overhead. Array partitioning involves dividing an array into multiple smaller partitions. This can be useful when optimizing memory accesses, cache utilization, and parallelism. By partitioning arrays, better data locality is achieved, and performance is improved by reducing memory bottlenecks. Dependence refers to the dependency between loop iterations, where the output of one iteration is required as input for the next iteration. By identifying and managing the loop-carried dependencies, code execution can be optimized, and performance is improved by enabling parallelism. Loop flattening is a technique that transforms nested loops into a single loop, reducing loop overhead and improving cache utilization. It eliminates the overhead associated with nested loops.

Additionally, the above-mentioned techniques optimize the hardware resource utilization and reduce the bottlenecks. Furthermore, these techniques exploit parallelism, allowing for concurrent execution of instructions or iterations. Despite the significant advantages, some potential drawbacks associated are the complexity and trade-offs involved in implementing and debugging these optimisations based on the violations in the HLS compiler. Hence, judicial use of optimisation methods aids the improvement in performance. So, the optimisation process was carried out incrementally. Firstly, individual directives were included independently to understand their influence on latency and hardware resources. Subsequently, multiple directives were combined to determine the best-performing implementation. The final optimized design corresponds to the combination of pipelining, loop unrolling, array partitioning, dependence removal, and loop flattening, as reported in Fig. 3.

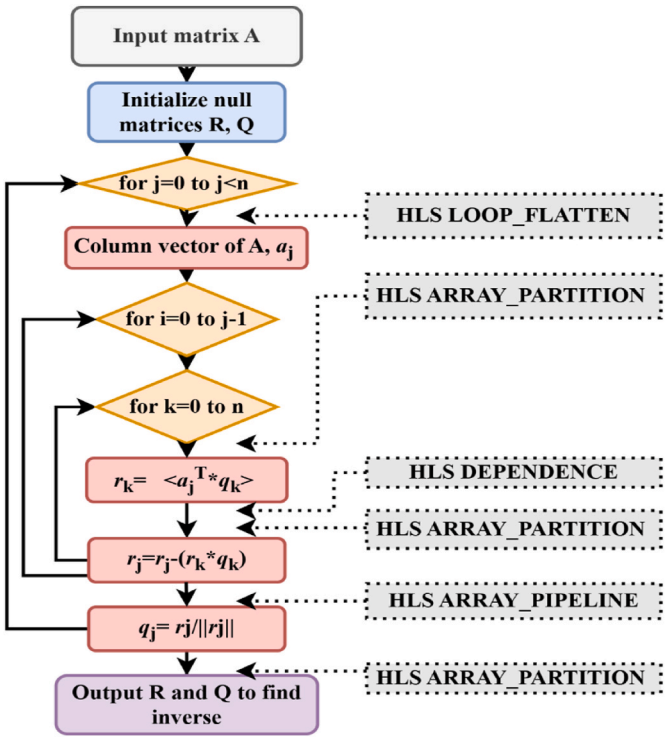


Fig. 3. Various optimisations introduced into the source algorithm.

4.3. Pragmas

While the HLS tool can automatically schedule, bind and allocate resources, their performance is limited by the properties of the given algorithm. The initial implementations of MGS-QRD and GE/IGE methods have loop-carried dependencies, loops with variable numbers of iterations and multiple accesses to the same elements of the matrix. This restricts the ability of the HLS tool to fully exploit parallelization due to the sequential nature of the loop iterations, as each of them depends on the result of its predecessors. Memory conflicts also prevent the concurrent execution of certain parts, thus increasing the latency. As a result, the naive implementation of the algorithms leads to conservative scheduling, increased initiation intervals, higher latency, and inefficient hardware resource utilization. In order to overcome this limitation, HLS tools provide several ways to improve performance, such as automatic optimisation, latency directives and pipelining for concurrent optimisation. The current work presents several directive-based optimizations, such as loop pipelining, loop unrolling, array partitioning, dependency optimisation and loop flattening, which allow for exploiting additional parallelism and improve scheduling without changing the algorithm itself.

Pragmas are directives that can be applied in the source code. They can change the synthesis results for all implementations of the code, thereby optimizing the design and improving the throughput performance. This especially applies to iterative control structures like the loops and loop nests that determine the design performance in terms of time and space complexity. Likewise, pragmas for pipelining, array partitioning, loop unrolling and eliminating dependencies optimize the latency and execution time. Loop optimisation involves reducing the number of clock cycles needed for execution by loop unrolling, loop pipelining, and loop fusion. Dataflow optimisation involves streamlining the data flow inside the design to enhance performance using strategies including data pipelining, data splitting, and data reuse.

Resource sharing helps in reducing the size and power usage of the architecture. The shared resources include the adders, multipliers, and memory blocks. Besides, to maximize resource use, strategies like common subexpression removal can also be employed. The matrix inversion algorithm involving the matrix factorisation requires hardware acceleration. There are two primary challenges in implementing these algorithms using HLS tools. Firstly, loop-carried dependencies or the iterations of the main loop rely on calculations done in previous iterations, and therefore, they cannot be pipelined or unrolled. Second, the inner loop has a non-constant number of iterations, leading to difficulty in unrolling. Hence, the pragmas were inserted only after verifying that they did not violate data dependencies or numerical correctness. Different combinations of directives were evaluated experimentally, and the directives listed in Table 1 and shown in Fig. 3 produced the best trade-off between latency and resource utilization. The sample source code for matrix generation, and matrix inversion using optimized MGS, indicating the directives used, will be available for reference at [27].

4.4. Experimental setup and performance metrics

The proposed experimental setup and configuration for implementing the matrix inversion algorithms are shown in Table 2. To ensure consistent interpretation of the reported performance, the principal hardware metrics used in this work are defined as follows. Table 3 shows

Table 1
HLS directives used in the proposed implementation.

HLS Directive	Location	Purpose
#pragma HLS PIPELINE II = 1	Column Normalization and Inner vector loops	Reduces initiation interval and improves throughput
#pragma HLS UNROLL off	Small inner loops performing vector operations	Exploits parallel arithmetic operations. It is evaded in the proposed optimized implementation to reduce hardware overhead.
#pragma HLS ARRAY_PARTITION complete	Matrix arrays storing Q, R and intermediate vectors	Splits BRAM matrix arrays into individual registers/ banks, enabling multi-port concurrent read/write operations; Enables simultaneous accesses of data
#pragma HLS DEPENDENCE false	Loops with removable loop-carried dependencies - Inner projection dot-product calculation	Instructs the HLS scheduler to ignore false loop-carried data dependencies across non-overlapping column operations, improving pipeline scheduling
#pragma HLS LOOP_FLATTEN	Nested loop structures (Outer loop nest)	Transforms nested outer loop boundaries into a single flattened loop, eliminating clock-cycle overhead from loop branch and control logic.

Table 2
FPGA-HLS implementation Configuration.

Parameters	Description
Programming Language	C/C++
HLS Tool	Xilinx Vitis/Vivado HLS
Target FPGA	Zynq XC7Z020-CLG484-1
Top Function	Matrix inversion
Functional Modules	Matrix factorisation → Upper triangular inversion → Final inverse computation
Clock Target	10 ns (100 MHz)
Verification	C Simulation + C/RTL Co-simulation
RTL Verification	Synthesis + Implementation
Test Matrices	Random non-singular matrices
Matrix Sizes	4×4, 10×10, 16×16, 25×25, 32×32
Number of Test Cases	100 matrices per configuration, 1000 matrices for Monte Carlo Analysis
Floating Point	IEEE-754 Single (32-bit), Double (64-bit)
Fixed Point	ap_fixed<32,16>, ap_fixed<64,32>
Performance Metrics	LUT, FF, BRAM, DSP, Latency, Fmax, Throughput

Table 3
Performance metrics.

Metrics	Formula	Interpretation
Clock Period	$T_{clk} = \frac{1}{f_{\max}}$	Duration of one clock cycle
Clock Cycles	N_{cycles}	Number of cycles required
Latency	$L = N_{cycles} \times T_{clk}$	Time for one matrix inversion
Throughput	$TP = \frac{1}{L} = \frac{f_{\max}}{N_{cycles}}$	Matrix inversions per second

the performance metrics. $f_{\max}$ denotes the maximum operating frequency; N_{cycles} is the number of clock cycles; T_{clk} is the clock period. Performance assessment of the hardware must use several timing metrics, instead of focusing only on the hardware resources. For this design, the maximum frequency (F_{max}) attained the number of clock cycles, the latency and the throughput are considered together since they all contribute to the performance efficiency of the designed FPGA architecture. While the target clock period is the timing constraint when synthesizing, the Fmax that is attained depends on the critical path in the hardware created.

To evaluate numerical accuracy, the computed inverse matrix was validated through matrix reconstruction. For each experiment, the product of the original matrix A and its computed inverse A^{-1} was compared with the identity matrix I. The inversion error was quantified using the normalized Frobenius norm:

$$E = \frac{\left\| AA^{-1} - I \right\|_F}{\left\| I \right\|_F} \tag{10}$$

where, $\| \cdot \|_F$ denotes the Frobenius norm. The normalized error is an indicator of how far the reconstructed identity matrix is from the expected outcome, and it is frequently employed in assessing the accuracy of numerical matrix inversion procedures. The experiment consists of evaluating one hundred randomly generated non-singular matrices for each tested matrix size and each type of number representation. The pseudo-random benchmark matrices were generated using a custom C testbench implementing a Uniform Distribution. The matrices were checked for non-singularity, and rank-deficient matrices were discarded and regenerated. A fixed seed of 1 was used. The same set of 100 matrices were used across all arithmetic representations (fixed-point, single-precision, and double-precision floating point) so that the comparison across representations is matched. For the 1000-matrix Monte Carlo analysis, the uniform generation mechanism is maintained, but the range is normalized to [-1.0, 1.0] to maintain well-conditioned matrix norms. The error indicated is the mean value of the normalized reconstruction error. Such an approach allows comparing the performance of fixed-point and floating-point methods consistently.

5. Results and discussions

In our work, initially, an experimental attempt to introduce different optimisations in floating point QRD by MGS is analysed and tabulated as shown in Table 4. It is an ablation study conducted on the single-precision floating-point MGS core illustrating the trade-offs and improvements attributed by pipelining, unrolling, partitioning, dependence handling, or their combination.

Later, the identified pragmas that evade violations and produce comparatively better utilization and timing are used in the matrix inversion as shown in Fig. 3. With reference to MGS, the five pragmas are incorporated at appropriate loops in the source code as optimisation strategies. They are pipelining, unrolling, array partitioning, dependence and loop flattening. The three Gram-Schmidt variants, namely, NGS, CGS and MGS and the GE, IGE are first implemented in the HLS tool. Factorisation of square matrices of different sizes, 4, 16, 100 and 500, was verified by simulation. For instance, Fig. 4 shows the timeline trace diagram by simulation of the synthesized MGS Algorithm. The RTL blocks of the design are shown in Fig. 5.

The execution profile reveals that there is no significant stalling across the loop executions. The design features the execution of loop modules, including pipelined loops, which indicate effective scheduling and optimisation. The time trace indicates the 3 major functional modules defined in the source code and executed sequentially. One for computing **Q** and **R** (decomposition), one for performing the inverse of **R** and finally computing the inverse of the original matrix **A** by multiplying with the transpose of **Q**.

Fixed-point and floating-point implementations at 64-bit precision and 32-bit precision were incorporated for comparison. Embedding pragmas and directives at appropriate places in the code using the HLS tool helps optimisation. Similarly, simulation and synthesis reports are analysed for GE methods. The optimisation methods impact the area and timing performance, and further throughput is calculated by dividing the maximum clock frequency (in MHz) by the number of clock cycles to perform matrix inversion.

Table 4
Implementation of HLS optimisation options in single precision floating point MGS-based QRD for a 10x10 matrix.

Parameters	No optimisation	Pipelining	Unrolling	Array Partitioning	Loop carried Dependence	Combination of Optimisations
Time period (ns)	5.069	6.582	5.156	5.312	5.328	5.138
SLICE	48	48	46	54	56	71
LUT	101	103	78	99	97	97
FF	167	164	172	164	158	233
SRL	1	2	0	1	2	1

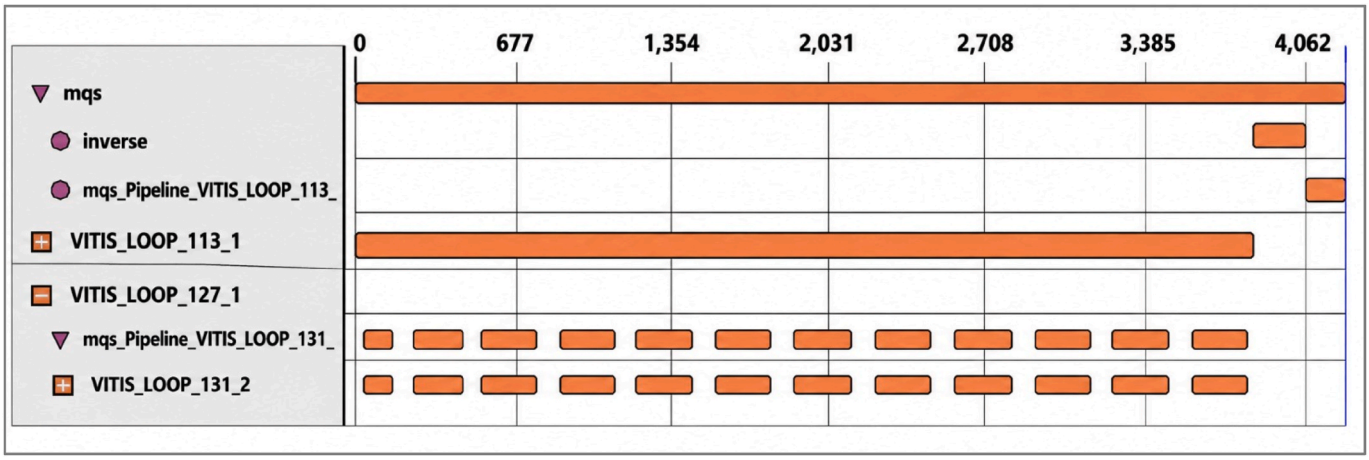


Fig. 4. Timeline trace diagram for the synthesized MGS algorithm.

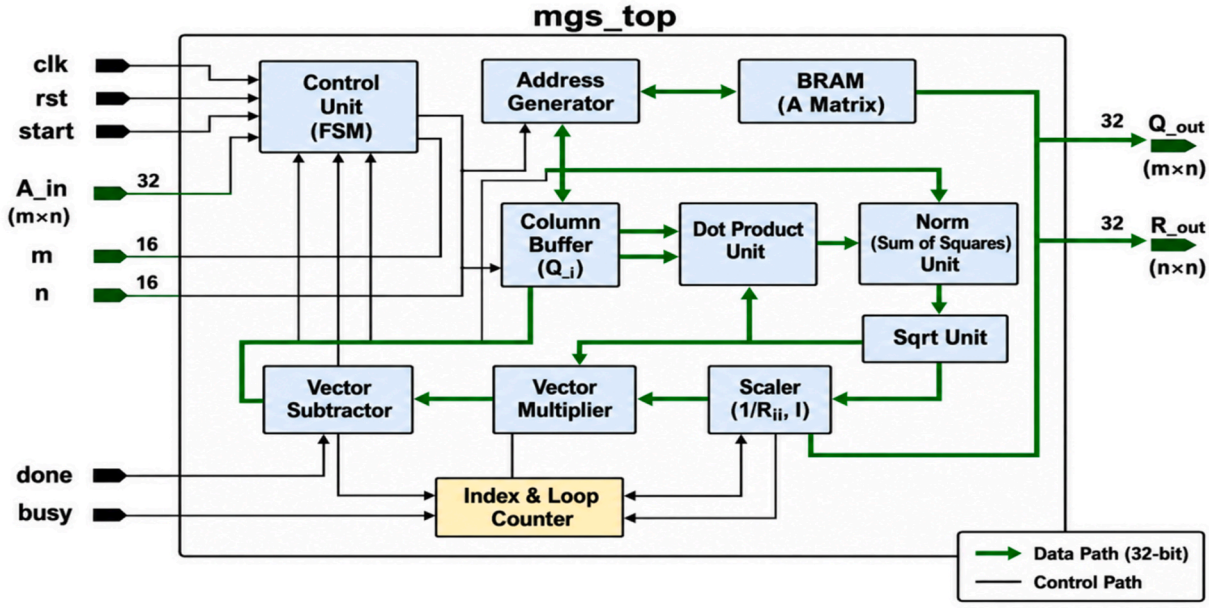


Fig. 5. RTL Block level Organisation of Proposed MGS in FPGA implementation.

5.1. Matrix inversion by QRD

QRD of a 10x10 matrix was performed for NGS, CGS and MGS algorithms. The results indicate that MGS requires reduced resources, as shown in Table 5. The proposed HLS models integrated with directives and pragmas to optimize the path have shown fewer utilization of slices and LUTs. The device considered for implementation is the Xilinx Zed board – Zynq xc7z020-clg484-1. The HLS implementation does not explicitly require Block RAMs because it does not require large memory structures or external access. Also, the reported usage of BRAM and DSP is relative to the resources calculated by the Vitis/Vivado HLS tool under the current configuration of the implementation process. The mapping of resources depends on synthesis parameters, compiler optimisation features, and the FPGA architecture. Accordingly, the usage of zero BRAM or zero DSP in the configurations under consideration means that the tool and optimisation directives have utilized the registers and internal LUT-based distributed logic resources fort mapping, rather than being implemented using BRAM and DSP. Fig. 6 compares the resource utilization graphically.

QRD is then integrated into the matrix inversion process. Matrix inversion is computed for 32-bit and 64-bit floating-point values in matrices of different sizes. Optimisation techniques were not adopted in

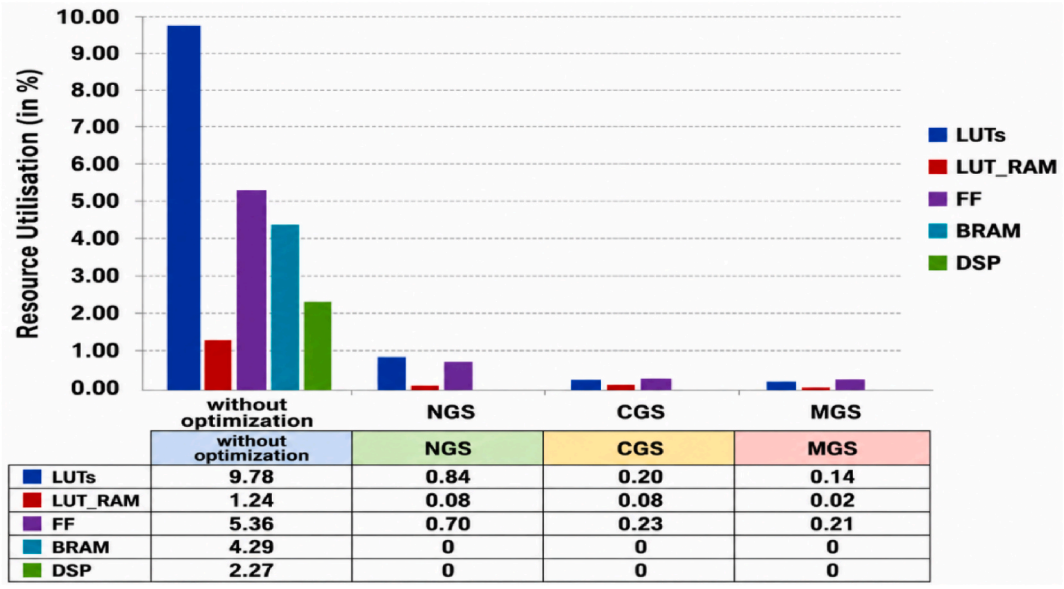
Table 5
Post RTL Implementation resource utilization of 10x10 matrix QRD.

Parameters	Available resources	Utilization for 32-bit floating point			
		without optimisation [15]	After optimisation		
			NGS	CGS	MGS
LUTs	53200	5203	447	106	74
LUT_RAM	17400	216	14	14	3
FF	106400	5703	745	245	223
BRAM	280	12	0	0	0
DSP	220	5	0	0	0

the initial study. Since MGS is found to utilize fewer resources, it is used for the inversion. Timing in ns and the resources utilized for different matrix sizes are tabulated as shown in Table 6. It is observed that there is a significant rise in the time and area with increasing matrix size. Optimisation techniques can be resorted to reduce time and resource utilization. After directive-driven optimisation as shown in Fig. 3, has been performed, it is observed in Table 7 that there is a trade-off between the reduction in time and the increase in slices, LUTs and FFs. The study also investigates the fixed-point notation. From Table 7 and it is notable that the overhead in fixed-point implementation is lower compared to the optimized floating-point implementations. Fixed-point operation requires 2% to 5% less time(ns) than a non-optimized floating-point operation. It also has lower resource utilization overhead.

32-bit fixed point showcased reduced timing as well as resource utilization than floating point. To ensure the suitability of 32-bit fixed-type data with 16-bit precision, investigations on errors in the resultant values are conducted. The input values of matrix elements are considered to be ranging between <-128, 127> for MGS-based matrix inversion and the observations are recorded in Table 8. For the 32-bit representation, the 16-bit integer component (inclusive of the sign bit) was selected to comfortably accommodate the input range of < -128,127> while providing headroom for increase in word-length during internal arithmetic. The remaining 16 bits were allocated to the fractional part, following a power-of-2 split that optimizes HLS multiplier efficiency. Experimental results in Table 8 reassure that this 16-bit fractional width is sufficient to maintain stability during reconstruction. Similarly <64,32> is chosen to be able to compare with double precision IEEE 754 format. Although floating point inversions yield the least amount of error, errors obtained in fixed point inversions are still acceptable for the majority of applications. For the error calculation, 100 random non-singular full-rank matrices were generated as sample matrices. The error is quantified by calculating the difference between the obtained product matrix (product of the original input and inverted output matrix) and an identity matrix to ensure effective matrix inversion. The error values obtained from the 100 matrices were accumulated, and the average error was computed. Hence the error reported in Table 8 is the average deviation of the reconstructed product matrix AA^{-1} from the corresponding identity matrix.

Besides hardware resource utilization, timing performance is an equally important metric in FPGA-HLS implementations. The proposed matrix inversion architectures were synthesized in Vitis HLS using a target clock period of 10 ns (100 MHz). The HLS synthesis report estimated a clock period of 8.76 ns, indicating that the scheduled design satisfies the specified timing constraint. The performance and resource estimate further provide latency (in clock cycles), initiation interval, pipeline status, and estimated resource utilization for each computational module, enabling early performance assessment during HLS synthesis. Although some floating-point computational loops exhibited small negative scheduling slack due to the complexity of IEEE-754



	without optimization	NGS	CGS	MGS
LUTs	9.78	0.84	0.20	0.14
LUT_RAM	1.24	0.08	0.08	0.02
FF	5.36	0.70	0.23	0.21
BRAM	4.29	0	0	0
DSP	2.27	0	0	0

Fig. 6. Comparison of resource utilization for 10x10 matrix QRD.

Table 6
Scalability of matrix sizes and post RTL implementation resource utilization for matrix inversion using MGS.

Parameters	Floating point (Single Precision-32 bit)					Floating point (double precision -64 bit)				
	4x4	10x10	16x16	25x25	32x32	4x4	10x10	16x16	25x25	32x32
clock period(ns)	3.678	5.011	5.484	5.547	5.610	5.036	5.000	5.740	5.062	5.367
SLICE	62	67	86	96	122	83	105	122	131	149
LUT	132	154	196	212	251	170	191	231	262	286
FF	207	254	312	357	417	310	380	455	521	587
SRL	4	4	4	4	4	5	5	5	5	5

Table 7
Post RTL implementation resource utilization for MGS-QRD in 10x10 matrix inversion.

Parameters	Fixed point			Floating point (Single Precision-32 bit)	
	<32,16> without optimisation	<32,16> Optimized	<64,32> Optimized	without optimisation	Optimized
clock period(ns)	5.071	4.867	4.733	5.011	4.473
SLICE	43	65	87	67	130
LUT	121	141	178	154	188
FF	167	193	265	254	527
SRL	1	7	9	4	2

Table 8
Resultant error performance comparison for MGS.

Data types	Floating point 32-bit	Floating point 64-bit	Fixed point <32,16>	Fixed point <64,32>
Error	0.000021	0.000001	0.000084	0.000012

arithmetic operations, this represents an intermediate scheduling estimate rather than the final implementation timing. Say, after RTL synthesis, placement, routing, and timing optimisation in Vivado, the implemented design for optimized MGS-QRD in 10x10 matrix inversion using 32 bit floating point data, achieved a post-implementation clock period of 4.473 ns, corresponding to an operating frequency of approximately 223.6 MHz. These observations demonstrate that the applied HLS optimisation directives, including loop pipelining, loop unrolling, array partitioning, and dependence optimisation, effectively improve scheduling efficiency, reduce critical-path delays, and enhance timing closure. Consequently, the proposed architectures achieve improved latency and throughput while maintaining efficient hardware resource utilization.

Future work will extend the numerical validation to ill-conditioned and near-singular matrices to further investigate the robustness and numerical stability of the proposed HLS-based implementations. However, the Monte Carlo validation in Table 9 demonstrates the robustness of the proposed matrix inversion architecture over 1000 randomly generated full-rank matrices. The mean and worst-case errors remain very small across all matrix sizes, while the success rate exceeds 99.5%, indicating reliable inversion performance and good numerical stability as the matrix dimension increases. The convergence plot in Fig. 7 shows that the average reconstruction error stabilizes as the number of Monte Carlo trials increases, confirming that the reported error statistics are statistically representative and are not influenced by a limited sample size. This addresses the reproducibility of the proposed implementation. The error histogram in Fig. 8 illustrates the distribution of reconstruction errors over all test matrices. The concentration of errors around very small values indicates consistent numerical accuracy, while the absence of significant outliers demonstrates the robustness of the proposed matrix inversion architecture across diverse input matrices.

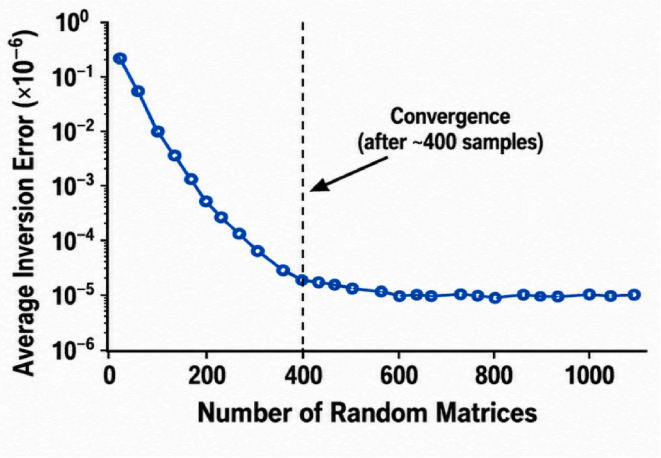


Fig. 7. Monte Carlo convergence analysis.

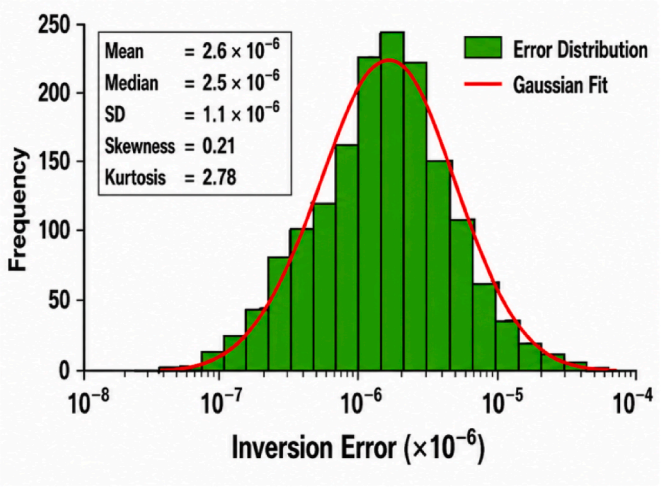


Fig. 8. Inversion matrix error histogram.

5.2. Matrix inversion by Gaussian Elimination method

HLS of GE method and IGE method for various matrix sizes is performed and is tabulated as shown in Table 10. It is inferred that an increase in matrix size does not largely affect the utilization of LUTs and FFs, but increases the requirement of BRAMs. Further, the synthesis results of the IGE method are analysed and presented in Table 11. When the floating point method was adopted, clock timing remained the same, but the resource utilization increased significantly by a factor of up to 2. In order to improve the efficiency, optimisation techniques are deployed for the GE methods also, and the reports are tabulated as (a) without optimisation and (b) with optimisation in Table 12. It is observed that optimisation improves timing slightly across most cases, and timing gains are modest. Optimisation benefits more in resource utilization than latency, as seen in the reduction of logic resource usage. Also, fixed-point implementations consume more LUTs and FFs than their floating-point counterparts. IGE is more resource-intensive than GE. GE methods prove to be more resource consuming and time consuming than QRD.

The generality of the proposed matrix inversion approach is demonstrated in Table 13, where the algorithms were assessed for

Table 9
Monte Carlo validation of 1000 random full-rank matrices.

Matrix Size	Mean Error	Worst Error	Success Rate (%)
4×4	1.8×10^{-6}	4.1×10^{-6}	100
10×10	2.0×10^{-6}	5.4×10^{-6}	100
16×16	2.6×10^{-6}	7.3×10^{-6}	99.8
25×25	3.4×10^{-6}	8.9×10^{-6}	99.7
32×32	4.2×10^{-6}	9.8×10^{-6}	99.5

Table 10
Scalability of matrix sizes and resource utilization for the GE Method.

Parameters	32-bit fixed-point					32-bit floating point				
	4x4	10x10	16x16	25x25	32x32	4x4	10x10	16x16	25x25	32x32
clock period(ns)	8.510	8.614	8.510	8.510	8.510	8.397	8.614	7.256	7.256	7.256
BRAM	2	4	4	6	8	0	2	2	4	4
LUT	4185	4276	4243	4427	4209	4926	3391	5019	5202	4989
FF	3227	3201	3227	3384	3291	2928	1582	2985	3142	3042
DSP48E	6	6	6	6	6	10	5	10	10	10

Table 11
Scalability of matrix sizes and resource utilization for IGE.

Parameters	32-bit fixed-point					32-bit floating point				
	4x4	10x10	16x16	25x25	32x32	4x4	10x10	16x16	25x25	32x32
clock period (ns)	8.510	8.510	8.510	9.634	8.510	8.397	9.634	8.397	9.634	7.256
BRAM	2	7	7	12	20	0	5	5	10	17
LUT	5220	6304	8070	12151	12975	5568	5621	5597	5682	5556
FF	4957	4780	4843	5212	5240	3470	3421	3460	3629	3495
DSP48E	9	9	9	12	9	10	11	10	14	10

Table 12
Resource utilization GE and IGE method in 10x10 matrix inversion.

Parameters	32-bit floating point				32-bit fixed point			
	GE		IGE		GE		IGE	
	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
clock period(ns)	8.614	8.434	9.634	9.634	8.614	8.326	8.510	8.325
SLICE	2	5	5	4	4	5	7	9
LUT	3391	2399	5621	4628	4276	6707	6304	8452
FF	1582	2314	3421	3932	3201	5896	4780	9805
SRL	5	5	11	11	6	7	9	12

Table 13
Scalability analysis of the proposed matrix inversion framework.

Matrix Size	Algorithm Verification	FPGA-HLS Detailed Analysis	Purpose
4 × 4	✓	✓	Representative hardware implementation
10x10	✓	✓	
16 × 16	✓	✓	Scalability verification
32 × 32	✓	–	Algorithm scalability evaluation
500 × 500	✓	–	Maximum evaluated matrix size

matrix dimensions starting from 4 × 4 up to 500 × 500. Nevertheless, the implementation results for FPGA-HLS of the algorithms are shown only for the representative matrix sizes that are used in MIMO systems. The representative matrix sizes are compared after implementation in terms of hardware, timing, and throughput with the previous related works [8–15,17] in the next section. GE and QRD based Least square solver were evaluated within a MATLAB implementation of OMP-based sparse signal reconstruction. The GE-based least-squares solver resulted in an RMSE of 0.5191 and a reconstruction SNR of 28.10 dB. In comparison, the QRD-based solver achieved a lower RMSE of 0.4503 and a higher reconstruction SNR of 34.6 dB, indicating more accurate sparse signal reconstruction. These results are consistent with the superior numerical stability of QR decomposition. When combined with the hardware results, the proposed QRD/MGS-based architecture offers an attractive balance of reconstruction accuracy, hardware efficiency, and low latency compared with the GE-based implementation. Since the least-squares computation constitutes the dominant operation in each OMP iteration, the proposed matrix inversion hardware can directly support this computational kernel. Improved reconstruction accuracy is expected to enhance channel-state information quality, which can subsequently benefit downstream signal detection and precoding. However, a complete communication-system-level evaluation, including BER/SER analysis using specific modulation schemes, antenna configurations, and MIMO detectors, is beyond the scope of the present work and is considered as future work.

5.3. Comparison with previous works

Although all proposed architectures were synthesized with a common target clock period of 10 ns, the achieved operating frequency varies according to the critical path generated after synthesis and implementation. The applied HLS directives reduce scheduling bottlenecks, improve operation overlap, and shorten the effective combinational path, enabling better timing closure. The maximum operating frequencies (F_{max}) reported in this work correspond to the post-implementation critical path and timing results obtained for each architecture. The comparative analysis with various works of FPGA-Based Matrix Inversion Implementations in the past has been done and their experimental Conditions are shown in Table 14 [11–14,18]. use systolic architectures and [12,14,16] focus on CORDIC structures to improve the performance. Even though pipelining and parallel processing prospects are used, the application of pragmas at appropriate places in the HLS of C/C++ codes seems to be an unexplored option. It is observed from Table 15, that GE methods discussed in this work have fairly better performance in terms of utilization (only 2399 LUTs, 2414 FFs, 4 BRAMs, and 4 DSPs for 4x4 matrix) compared to Gauss-Jordan in Ref. [10] for the same matrix size. Yet, the incremental way lowers the throughput and requires more cycles.

Moreover, the proposed HLS-based optimisation introduced in the MGS QRD has significantly reduced the utilization. It is observed that

Table 14
Comparison of experimental conditions for FPGA-Based matrix inversion implementations.

Ref.	Matrix Inversion/ Decomposition Algorithm	Arithmetic Type	Design Methodology	Performance Metrics
[5]	OMP computation-IGE	Fixed-point	RTL FPGA	Throughput, Hardware Resources
[8]	Matrix inversion-Gauss-Jordan Elimination	Floating-point	RTL Hardware	LUT, FF, Latency
[9]	Systolic Array QRD	Floating-point	RTL Systolic Architecture	Throughput, Frequency, Hardware Resources
[10]	GR-QRD	Floating-point	RTL Hardware	LUT, DSP, Latency
[11]	Matrix Inversion QRD + Piecewise Polynomial Approximation	Floating-point	Systolic Array	LUT, DSP, Frequency
[12]	QRD High-Throughput	Floating-point	RTL Pipeline	Throughput, Frequency, LUT
[13]	GS-QRD	Floating-point	HLS	LUT, DSP, Latency
[14]	GR unit	Floating-point	RTL Hardware	LUT, Frequency
[15]	CORDIC-Based - QRD	Fixed-point	RTL CORDIC Architecture	LUT, DSP, Throughput
[16]	Matrix Inversion, Hybrid Piecewise Polynomial Systolic Array	Floating-point	RTL Systolic Architecture	LUT, DSP, Frequency
[17]	Matrix Inversion, Floating-Point Matrix Inversion	Floating-point	RTL Hardware	LUT, DSP, Latency
[18]	OMP Accelerator	Fixed-point	RTL Pipeline	Throughput, Frequency, Hardware Resources
Proposed	**Optimized MGS-QRD + GE/IGE**	**Fixed-point, Floating point (32,64)**	**FPGA-HLS (Vitis/ Vivado HLS)**	**LUT, FF, DSP, BRAM, Fmax, Clock Cycles, Latency, Throughput**

BRAMs and DSPs are not used. However, a trade-off in terms of latency and throughput is observed. The error rate being negligibly small is an encouraging outcome. The optimized MGS QRD-based inversion for fixed point and floating point requires 4.867 ns and 4.473 ns clock period, respectively. The proposed optimisation directives for an Optimized MGS QRD for Matrix Inversion (32-bit floating-point and fixed-point) show low resource utilization (188–141 LUTs and 527–193 FFs), low latency (0.814–2.468 μs), and throughputs of 1.23×10^6 and 0.405×10^6 matrices/s, respectively, for 10×10 matrix computation. Comparing the optimized MGS floating-point design with the fixed-point version, the fixed-point implementation reduces LUT utilization by approximately 25% (141 vs. 188 LUTs) and FF utilization by approximately 63% (193 vs. 527 FFs) compared to the floating-point implementation. Although the fixed-point implementation occupies fewer hardware resources, it requires a larger number of execution cycles (507 versus 182), resulting in a higher latency (2.468 μs versus 0.814 μs) and lower throughput (0.405×10^6 versus 1.23×10^6 matrices/s) than the floating-point implementation. The floating-point implementation also achieves a higher maximum operating frequency (223.56 MHz versus 205.46 MHz), making it more suitable for applications requiring high computational throughput and real-time processing.

The proposed optimisation directives for an Optimized MGS QRD for Matrix Inversion (32-bit float and fixed) are competitive with earlier works. In comparison, the floating-point Gram-Schmidt QRD in Ref. [15] consumes significantly more resources with higher latency, though it offers higher throughput. Hence, the proposed designs provide an effective trade-off between hardware efficiency and timing performance, making them suitable for resource-constrained as well as high-throughput FPGA implementations. Moreover, the proposed design achieves better performance with significantly lower overhead, even for a larger 10×10 matrix, outperforming several earlier works that demonstrated results only for smaller 4×4 matrices. Other previous works, such as [13] and [16], emphasize systolic and CORDIC-based architectures. While they achieve high throughput, these designs typically incur greater complexity and are more demanding in terms of resources. Proposed strategy of deploying appropriate directives in the standard HLS flow, enabled better exploitation of concurrency in the algorithm, contributing to the improvements.

6. Conclusion

This work has demonstrated the implementation of matrix inversion using NGS, CGS and MGS algorithms for QRD and GE using HLS. The paper investigates its performance for various matrix sizes. Optimisation techniques through directives are used to reduce the latency and increase the throughput. The results demonstrate that careful application of HLS directives can lead to efficient hardware designs with lower development effort. The algorithms are synthesized for different data types, such as fixed-point and floating-point with 32-bit precision and 64-bit precision. Results of implementing matrix inversion indicate clear trade-offs between hardware resource utilization, timing performance, and computational throughput for fixed-point and floating-point implementations. Compared to the fixed-point implementation, the optimized floating-point implementation achieves an 8.8% higher maximum operating frequency, 67% lower latency, approximately three times higher throughput, and 75% lower numerical error, whereas the fixed-point implementation reduces LUT and FF utilization by approximately 25% and 63%, respectively. The optimized designs are scalable, applicable for integration into constrained embedded systems and for processing large datasets in parallel. There is scope for implementing and testing the optimized model for MIMO signal reconstruction applications using the generated IP blocks.

A limitation of the current research is the use of randomly generated non-singular matrices with well-conditioned numbers to perform the numerical computations. The proposed implementation of the FPGA-HLS was mainly done to analyse the hardware efficiency, latency, throughput, and numerical precision under reasonable operating conditions. Future work will focus on investigating the adaptivity of the fixed-point scaling, robustness to ill-conditioned and nearly singular matrices and arithmetic overflows. The improved FPGA-HLS implementations are well-suited for reconstructing through CS, and for real-time signal processing that calls for fast inversion of matrices. Since matrix inversion is one of the most frequently used operations in MIMO communication systems further work will involve incorporating the developed hardware in a fully-fledged system design and performance measurement through communication parameters. Although the new FPGA-HLS approach proves to be efficient in terms of hardware efficiency and performance, the current research only performs synthesis-level evaluations. Further research will consider the evaluation of the implementation stages such as place and route analysis, timing closure and power consumption.

Contributions

All authors are prepared the article. N.S and K.R: Reviewing, Supervision, Validation, Investigation, Methodology, Software, Editing, Methodology, Software.

Table 15
Comparison with previous works.

	Device	bit width	Data type	Slices	LUT	FF	BRAM	DSP	Latency (s)	Cycles	Matrix size	Fmax (MHz)	Throughput b (bits/s) or M (matrices/s)	Error
GUSTO Core Generator [9]	Virtex 4	20	fixed	3584	-	-	1	12	-	-	4x4	-	0.26e6 M	-
Gauss-Jordan [10]	Virtex 5	32	float	-	8549	-	-	22	-	-	4x4	301.4	-	-
Systolic array QRD [11]	XC4 VFX60-11	16	fixed	5389	9810	5891	-	41	0.35	5	4x4	144.4	28e6 M	-
Given rotation and CORDIC [12]	Virtex 5	16	float	10457	10899	-	-	28	4.062 n	180	4x4	246	24e6 M	4.2e-3
Systolic array polynomial approximation [13]	Virtex 5	24	fixed	1986	8705	1063	8	100	108.195 n	-	4x4	-	1.79 Gb	-
CORDIC Core Generator [14]	Virtex 5	16	fixed	-	2671	-	-	12	0.20 u	52	4x4	254	31.70 M	5.8e-4
Pipelined MGS QRD [8]	Virtex 5	16	complex	6173	4484		2	0	-	-	4x4	132.8	44.27 M	-
Floating point Gram-Schmidt-QRD [15]	Zynq	32	float	-	5862	4250	2	6	209 m	21073	10x10	-	-	-
CORDIC QRD for MIMO [17]	Altera Stratix 3	21	float	-	9834	-	-	0	0.39 u	64	4x4	164.6	20.57 M	4.7e-3
GR, Hub FP QRD [16]	Virtex 5	32	float	50547	-	-	0	52	-	296	7x7	287.8	41.11M	-
Systolic array-based QRD [18]	Virtex 5	16	float	-	1458	-	-	52	171 n	12	-	70	5.83 M	-
LDL Factorisation [5]	Virtex 7	32	float		250681		834	1839	2359 n	NR	72x72	250	589.1 M	-
Synthesis of GE for matrix inversion (proposed)	Zynq	32	fix	-	2399	2414	4	4	2.016u	229	4x4	113.58	0.49e6 M	-
Synthesis of IGE for matrix inversion (proposed)	Zynq	32	float	-	6707	5896	8	28	11.797 u	1078	4x4	91.38	0.085e6 M	-
Optimized MGS QRD for Matrix Inversion (proposed)	Zynq	32	float	130	188	527	0	0	0.814u	182	10x10	223.56	1.23M	2.1e-5
	Zynq	32	fixed	65	141	193	0	0	2.468u	507	10x10	205.46	0.405M	8.4e-5

Availability of data and materials

The datasets used and analysed during the current study are available from the Corresponding author upon reasonable request.

Funding declaration

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We wish to acknowledge and express sincere gratitude to the management of PSG Institutions for providing the VLSI Laboratory facilities and to all those who supported throughout the course of this research.

Data availability

Data will be made available on request.

References

[1] R. Manchanda, K. Sharma, A review of reconstruction algorithms in compressive sensing, in: IEEE International Conference on Advances in Computing, Communication & Materials (ICACCM), 2020, pp. 322–325.
[2] Y. Arjoune, N. Kaabouch, H. El Ghazi, A. Tamtaoui, A performance comparison of measurement matrices in compressive sensing, Int. J. Commun. Syst. 31 (2018) e3576.
[3] X.W. Zhang, L. Zuo, M. Li, J.X. Guo, High-throughput FPGA implementation of matrix inversion for control systems, IEEE Trans. Ind. Electron. 68 (2020) 6205–6216.
[4] S. Roy, D.P. Acharya, A.K. Sahoo, Low-complexity architecture of orthogonal matching pursuit based on QR decomposition, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 27 (2019) 1623–1632.
[5] S. Roy, D.P. Acharya, A.K. Sahoo, Incremental Gaussian elimination approach to implement OMP for sparse signal measurement, IEEE Trans. Instrum. Meas. 69 (2019) 4067–4075.
[6] A. Alhamed, S. Alshebeili, FPGA implementation of complex-valued QR decomposition, in: IEEE 5th International Conference on Electronic Devices, Systems and Applications (ICEDSA), 2016, pp. 1–4.

[7] A. Irturk, B. Benson, S. Mirzaei, R. Kastner, An FPGA design space exploration tool for matrix inversion architectures, in: IEEE Symposium on Application Specific Processors, 2008, pp. 42–47.
[8] J. Arias-García, R.P. Jacobi, C.H. Llanos, M. Ayala-Rincón, A suitable FPGA implementation of floating-point matrix inversion based on Gauss-Jordan elimination, in: IEEE VII Southern Conference on Programmable Logic (SPL), 2011, pp. 263–268.
[9] M. Abels, T. Wiegand, S. Paul, Efficient FPGA implementation of a high throughput systolic array QR-decomposition algorithm, in: IEEE Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR), 2011, pp. 904–908.
[10] S. Aslan, S. Niu, J. Saniie, FPGA implementation of fast QR decomposition based on givens rotation, in: IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS), 2012, pp. 470–473.
[11] L.C. Santos, A.C. Atoche, J.V. Castilloy, O.L. Gandaraz, R.C. Alvarez, J.O. Aguilar, An improved hardware design for matrix inverse based on systolic array QR decomposition and piecewise polynomial approximation, in: International Conference on Reconfigurable Computing and Fpgas (ReConFig), 2015, pp. 1–6.
[12] S.D. Muñoz, J. Hormigo, High-throughput FPGA implementation of QR decomposition, IEEE Trans. Circuits Syst. II Express Briefs 62 (2015) 861–865.
[13] P. Desai, S. Aslan, J. Saniie, FPGA implementation of gram-schmidt QR decomposition using high level synthesis, in: IEEE International Conference on Electro Information Technology (EIT), 2017, pp. 482–487.
[14] J. Hormigo, S.D. Muñoz, Efficient floating-point givens rotation unit. Circuits syst, Signal Process. 40 (2021) 2419–2442.
[15] H.T. Nguyen, X.T. Nguyen, T.T. Hoang, C.K. Pham, A CORDIC-based QR decomposition for MIMO signal detector, IEICE Electron. Express 15 (2018) 20180174.
[16] J. Vázquez-Castillo, A. Castillo-Atoche, R. Carrasco-Alvarez, O. Longoria-Gandara, J. Ortegón-Aguilar, FPGA-based hardware matrix inversion architecture using hybrid piecewise polynomial approximation systolic cells, Electronics 9 (2020) 182.
[17] S. Chetan, J. Manikandan, V. Lekshmi, S. Sudhakar, Hardware implementation of floating-point matrix inversion modules on FPGAs, in: IEEE 32nd International Conference on Microelectronics (ICM), 2020, pp. 1–4.
[18] V.Q. Nguyen, W.H. Son, M. Parfieniuk, L.T. Trung, S.Y. Park, High-throughput and low-area implementation of orthogonal matching pursuit algorithm for compressive sensing reconstruction, ETRI J. 42 (2020) 376–387.
[19] D.C. Lay, Linear Algebra and Its Application, fifth ed., Pearson, 2016.
[20] D. Tse, P. Viswanath, Fundamentals of Wireless Communication, Cambridge University Press, 2005.
[21] V.H.P. Reddy, et al., FPGA enabled ECG signal reconstruction based on an enhanced orthogonal matching pursuit algorithm, Microprocess. Microsyst. 101, (2025) 102311.
[22] S. Sirois, et al., High level synthesis strategies for ultra fast and low latency matrix inversion implementation for massive MIMO processing, Integration, 82 (2022, 29-36).
[23] A.E. Baye, Performance analysis of channel estimation techniques for IRS assisted MIMO, Sci. Rep. 13 (1) (2023) 13574.
[24] R.C. Chang, C.H. Lin, K.H. Lin, C.L. Huang, F.C. Chen, Iterative QR decomposition architecture using the modified gram–schmidt algorithm for MIMO systems, IEEE Trans. Circuits Syst. I Regul. Pap. 57 (2010) 1095–1102.
[25] K.V. Kumar, V.R. Kopparthi, S.L. Sabat, R. Peesapati, System on chip implementation of floating point matrix inversion using modified gram-schmidt based QR decomposition on PYNQ FPGA, in: IEEE International Symposium on Smart Electronic Systems (Ises), 2021, pp. 84–88.
[26] J.L. Stanislaus, T. Mohsenin, High performance compressive sensing reconstruction hardware with QRD process, in: IEEE International Symposium on Circuits and Systems (ISCAS), 2012, pp. 29–32.
[27] Susithra N, "Matrixinversion: MGS-QRD and GE Based Matrix Inversion in HLS," GitHub repository. [Online] https://github.com/susithran/matrixinversion.